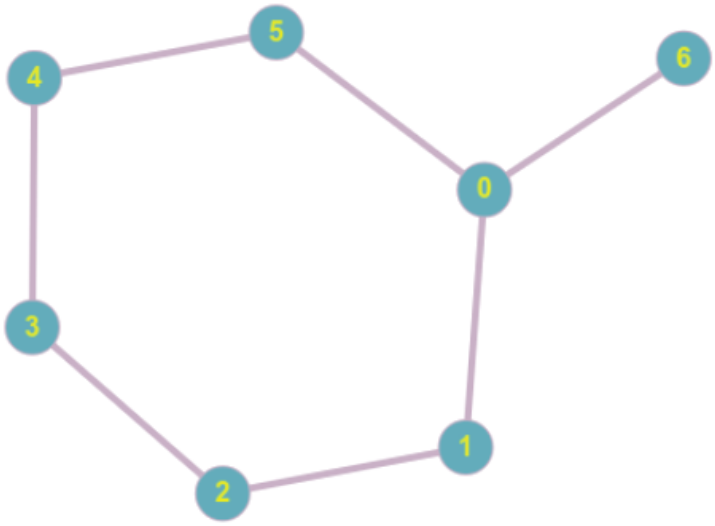
MSBD 5008 Homework 1

Pranav A, 20478966

1 Bridges

From the adjacency matrix, the following graph is plotted.



1. If the edge between 0 and 6 are removed, the graph is completely disconnected. Hence, 0-6 is a bridge.
2. The endpoints of 4-5 have no "friends" in common. If that edge is removed, the shortest path between them is of 5 edges. Hence 4-5 is a local bridge with a span of 5.
3. Similarly, as the previous point, 0-1, 1-2, 2-3, 3-4, 5-0 are local bridges with a span of 5.

2 Degree and Edge Distribution Simulations

2.1 Simulation Methodology

Following methodology is adopted for the emperical distribution verification.

1. **Emperical Distribution:** The graph is simulated for n number of times (depends on computing power and our patience). A running list keeps track of degrees or edges which are generated. Finally, that list is converted into the frequency list, then into a probabilistic list.

2. **Theoritical Distribution:** From the formulae given in the slides, they are simulated in order to get the expected values from the distribution.

3. **Graphical Verification:** The two distributions are plotted. Based on proximity, one could judge whether the emperical distribution conforms to theoritical distribution.

4. **Graphical Verification with correaltion plot:** Other way for confirming, is to plot a scatter plot between emperical and theoritical distribution. If it lies on a straight line, it means that they are well correlated.

5. **Statistical Pearson-R test:** Correlation test is applied on these distributions for verification. If the correlation coefficient r is closer to 1 and p-value is less than 0.05, it could be said that emperical distribution is verified as the theoritical distribution.

2.2 Simulations for a small random graph

The relevant libraries are imported.

```
import collections
import networkx as nx
import matplotlib.pyplot as plt
from scipy.special import comb
import scipy.stats as stats
import numpy as np
```

2.2.1 Edge Distribution Emperical Simulation

After that, graph is simulated 100000 times. Every loop generates the random graph and the number of edges are stored in a cumulative list.

```
simulations = 100000

count = 0
edge_list = [] # Keep track of the number of edges generated
while count < simulations:
        g = nx.gnp_random_graph(n = 20, p = 0.4, directed=False)
        edge_list.append(len(g.edges()))
        count += 1
```

From the results of the cumulative list, the frequency list is generated. These frequencies are converted to probabilities to result in a edge distribution.

```
counter = collections.Counter(edge_list)
edge_num = list(counter.keys())
freq = list(counter.values())
edge_prob = [i / sum(freq) for i in freq]

edge_num = edge_num[2:53]
edge_prob = edge_prob[2:53]
```

2.2.2 Edge Distribution Theoritical Simulation

Once the emperical distribution is calculated, the theoritical distribution needs to be calculated. The edge distribution $P(E)$ is given by:

$$P(E) = \binom{E_{max}}{E} p^E (1-p)^{E_{max}-E}$$

where $E_{max} = \binom{n}{2}$, which can be implemented in NumPy:

```
def p_e(n, p, e):
        return np.multiply(comb(comb(n, 2), e, exact=True),
        np.multiply(np.power(p, e),
        np.power((1 - p), (comb(n, 2) - e))))
```

It's important to implement this in one line in order to avoid precision loss in floating points. Now, this can be implemented on the range of edges.

```
exp_counts = [p_e(20, 0.4, i) for i in edge_num]
```

2.2.3 Distribution Comparision

The graph is plotted to visualize the edge distributions. The discrete distribution is plotted as the bar chart and theoritical distribution is plotted as the line chart.

```
plt.bar(edge_num, edge_prob, color='red')
plt.plot(edge_num, exp_counts)
plt.show()
```

The result is shown in Figure 1.

For verification, scatter plot is plotted of emperical and theoritical distribution.

The scatter plot reveals that emperical and theoritical distribution are linearly correlated.

Finally, applying the Pearson R test on these distribution:

```
stats.pearsonr(edge_prob, exp_counts)
```

This outputs the correlation coefficient as 0.99970317 and p-value as `1.0105400442849781e-80`. The small p-value indicates that the emperical distribution is indeed a binomial distribution.

Figure 1: Emperical vs Theoritical Distribution of Edges

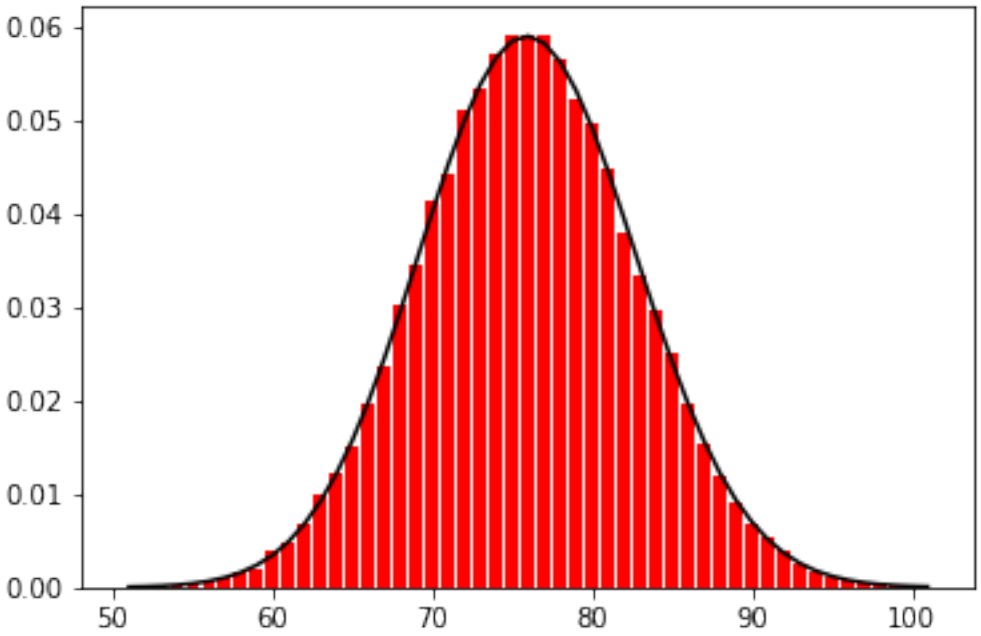
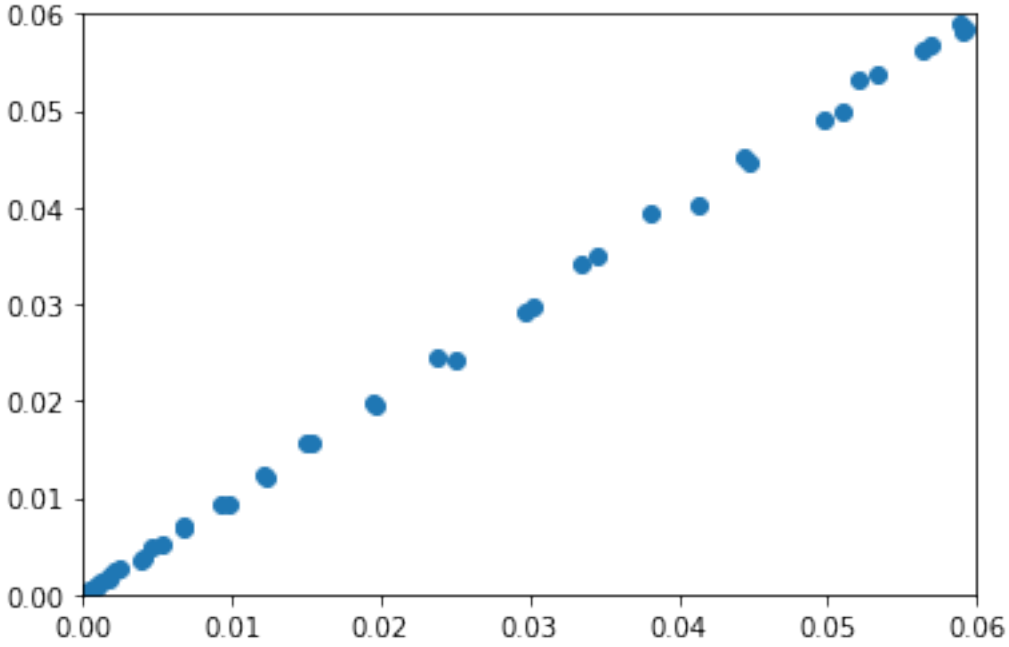


Figure 2: Correlation Plot of Emperical vs Theoritical Distribution of Edges



2.2.4 Degree Distribution Emperical Simulation

The graph is simulated 100000 times. Every loop keeps the track of degree information which was generated.

```
i = 0
deg_list = []
while i < 100000:
        g = nx.gnp_random_graph(n = 20, p = 0.4, directed=False)
        deg_list.append(list(zip(*list(g.degree())))[1])
        i += 1
```

The degree are converted into the frequency list, and then into the probabilities.

```
flatten = [item for sublist in deg_list for item in sublist]
counter = collections.Counter(flatten)
degrees = list(counter.keys())
freq = list(counter.values())
deg_prob = [i / sum(freq) for i in freq]
```

2.2.5 Degree Distribution Theoritical Simulation

Once the emperical distribution is calculated, the theoritical distribution needs to be calculated. The edge distribution $P(E)$ is given by:

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

The NumPy implementation is given by:

```
def p_k(n, p, k):
        return comb(n - 1, k) * (p ** k) * ((1 - p) ** (n - 1 - k))
```

Now, this can be implemented on the range of edges.

```
exp_deg = [p_k(20, 0.4, k) for k in degrees]
```

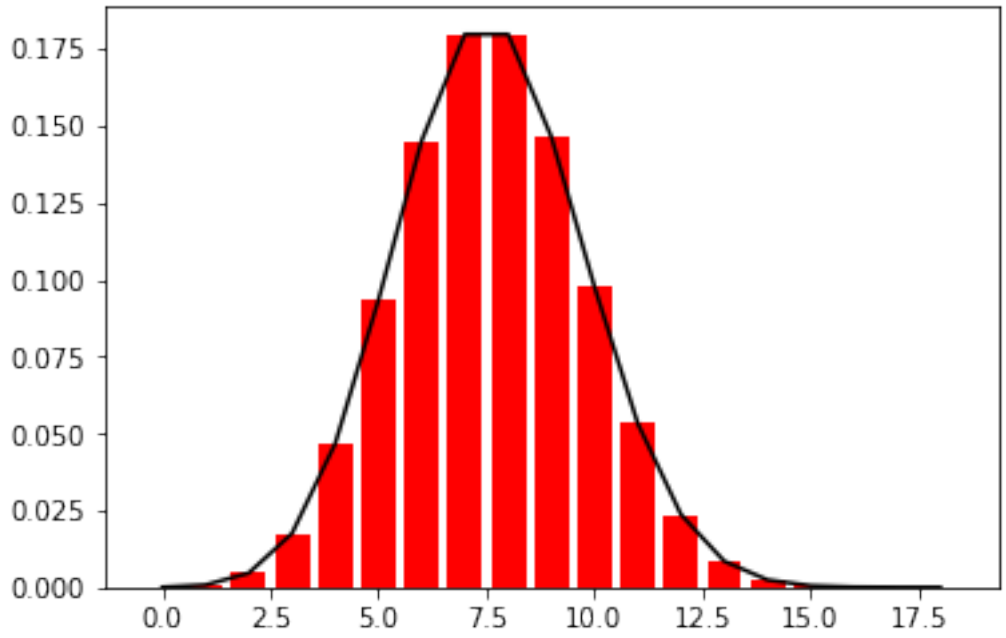
2.2.6 Distribution Comparision

The graph is plotted to visualize the degree distributions. The discrete distribution is plotted as the bar chart and theoritical distribution is plotted as the line chart.

```
plt.bar(degrees, deg_prob, color='red')
plt.plot(degrees, exp_deg, color='black')
plt.show()
```

The result is shown in Figure 3.
For verification, scatter plot is plotted of emperical and theoritical distribution.

Figure 3: Emperical vs Theoritical Distribution of Degrees




```
plt.scatter(deg_prob, exp_deg)
plt.xlim(0, 0.2)
plt.ylim(0, 0.2)
plt.show()
```

The scatter plot reveals that emperical and theoritical distribution are linearly correlated. Finally, applying the Pearson R test on these distribution:

```
stats.pearsonr(deg_prob, exp_deg)
```

This outputs the correlation coefficient as 0.999996 and p-value as `1.689070392743858e-45`. The small p-value indicates that the emperical distribution is indeed a binomial distribution.

2.3 Simulations for a big random graph

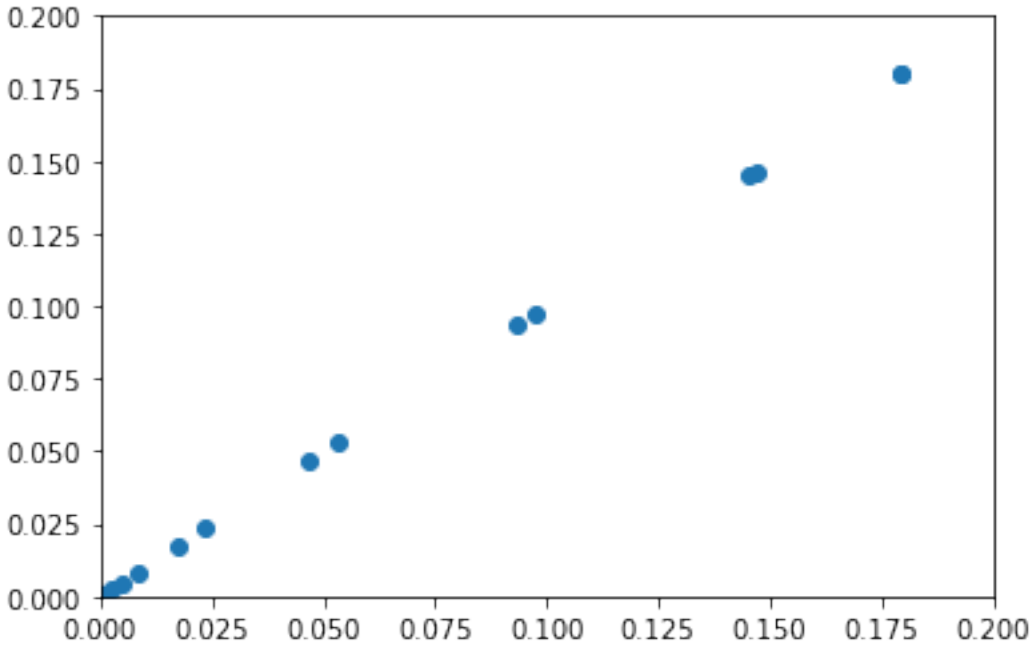
2.3.1 Edge Distribution Emperical Simulation

Graph is simulated 100000 times. Every loop generates the random graph and the number of edges are stored in a cumulative list.

```
simulations = 10000

count = 0
edge_list = [] # Keep track of the length of edges generated
while count < simulations:
        g = nx.gnp_random_graph(n = 200, p = 0.4, directed=False)
```

Figure 4: Correlation plot of Emperical vs Theoritical Distribution of Degrees




```
        edge_list.append(len(g.edges()))
        count += 1
```

From the results of the cumulative list, the frequency list is generated. These frequencies are converted to probabilities to result in a edge distribution.

```
counter = collections.Counter(edge_list)
edge_num = list(counter.keys())
freq = list(counter.values())
edge_prob = [i / sum(freq) for i in freq]
```

2.3.2 Edge Distribution Theoritical Simulation

Once the emperical distribution is calculated, the theoritical distribution needs to be calculated. The edge distribution $P(E)$ is given by:

$$P(E) = \binom{E_{max}}{E} p^E (1-p)^{E_{max}-E}$$

where $E_{max} = \binom{n}{2}$.

However, binomial coefficients of $\binom{200}{2}$ are too big to compute for a 64-bit computer. It is known that, as the numbers get larger, the binomial distribution approaches towards the normal distribution. Hence,

$$\mu = np$$

$$= \binom{200}{2} \cdot 0.4$$
$$\sigma = \sqrt{np(1-p)}$$
$$= \sqrt{\binom{200}{2} \cdot 0.4 \cdot 0.6}$$
$$P(E) = \mathcal{N}(E|\mu, \sigma^2)$$

which can be implemented in NumPy:

```
def p_e_large(n, p, e):
        n = comb(n, 2)
        mean = n * p
        std = (n * p * (1 - p)) ** 0.5
        return stats.norm(mean, std).pdf(e)
```

It's important to implement this in one line in order to avoid precision loss in floating points. Now, this can be implemented on the range of edges.

```
exp_counts = [p_e_large(200, 0.4, i) for i in edge_num]
```

2.3.3 Distribution Comparision

The graph is plotted to visualize the edge distributions. The discrete distribution is plotted as the bar chart and theoritical distribution is plotted as the line chart.

```
plt.bar(edge_num, edge_prob, color='red')
plt.plot(edge_num, exp_counts)
plt.show()
```

The result is shown in Figure 6.

For verification, scatter plot is plotted of emperical and theoritical distribution.

The scatter plot reveals that emperical and theoritical distribution are linearly correlated.

Finally, applying the Pearson R test on these distribution:

```
stats.pearsonr(edge_prob, exp_counts)
```

This outputs the correlation coefficient as 0.9714250 and p-value as `1.0335461385061103e-260`. The small p-value indicates that the emperical distribution is indeed a binomial distribution.

2.3.4 Degree Distribution Emperical Simulation

The graph is simulated 100000 times. Every loop keeps the track of degree information which was generated.

```
i = 0
deg_list = []
while i < simulations:
        g = nx.gnp_random_graph(n = 200, p = 0.4, directed=False)
```

Figure 5: Emperical vs Theoritical Distribution of Edges

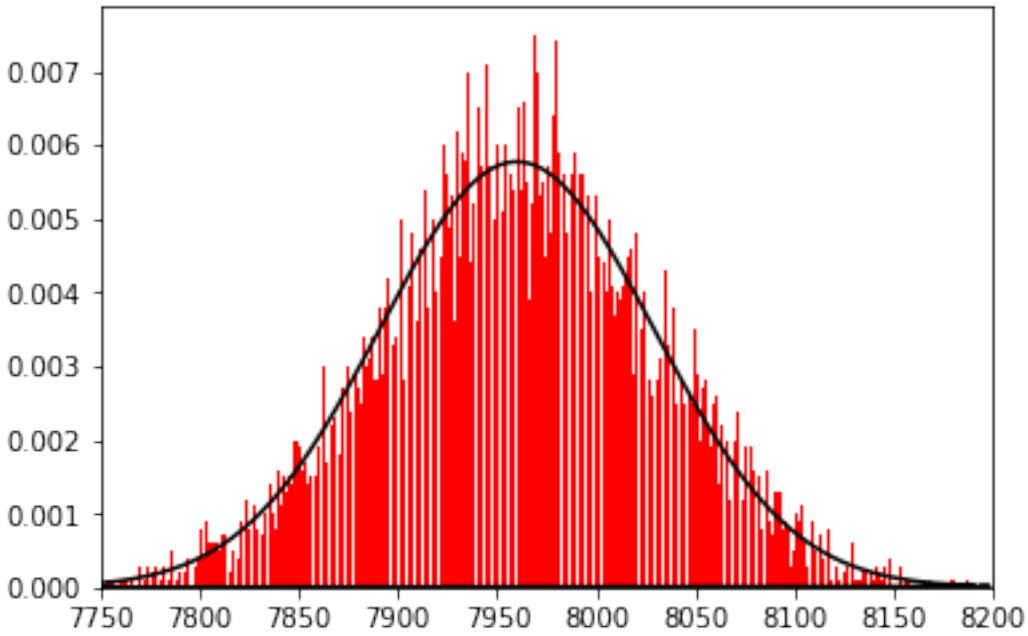
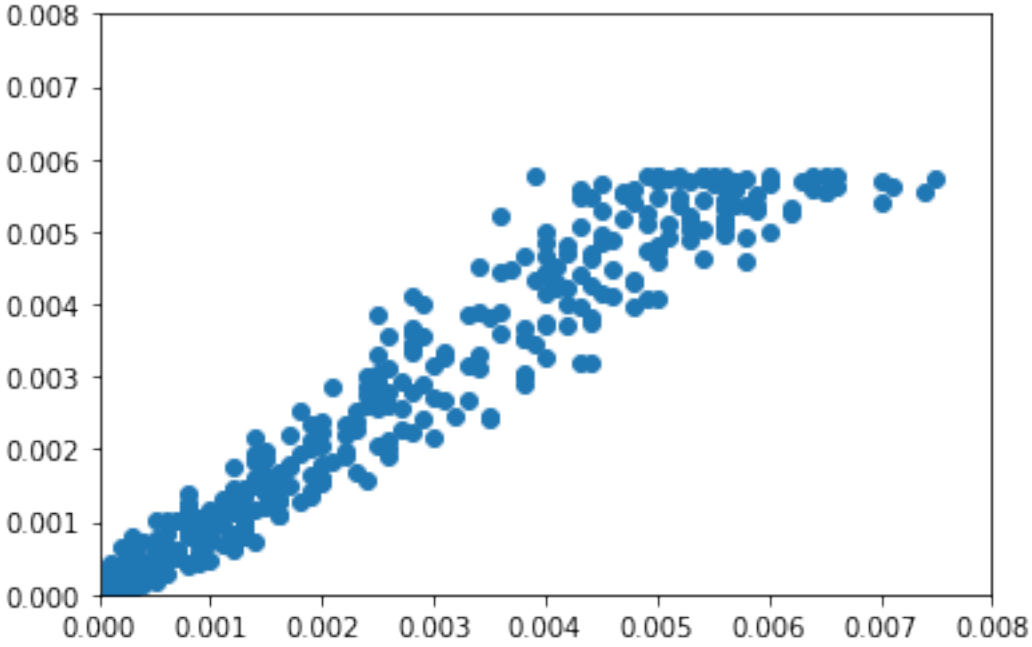


Figure 6: Correlation Plot of Emperical vs Theoritical Distribution of Edges



```
        deg_list.append(list(zip(*list(g.degree())))[1])
        i += 1
```

The degree are converted into the frequency list, and then into the probabilities.

```
flatten = [item for sublist in deg_list for item in sublist]
counter = collections.Counter(flatten)
degrees = list(counter.keys())
freq = list(counter.values())
deg_prob = [i / sum(freq) for i in freq]
```

2.3.5 Degree Distribution Theoritical Simulation

Once the emperical distribution is calculated, the theoritical distribution needs to be calculated. The edge distribution $P(E)$ is given by:

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

The NumPy implementation is given by:

```
def p_k(n, p, k):
        return comb(n - 1, k) * (p ** k) * ((1 - p) ** (n - 1 - k))
```

Now, this can be implemented on the range of edges.

```
exp_deg = [p_k(200, 0.4, k) for k in degrees]
```

2.3.6 Distribution Comparision

The graph is plotted to visualize the degree distributions. The discrete distribution is plotted as the bar chart and theoritical distribution is plotted as the line chart.

```
plt.bar(degrees, deg_prob, color = 'red')
plt.plot(degrees, exp_deg, color = 'black')
plt.xlim(55, 105)
plt.show()
```

The result is shown in Figure 7.

For verification, scatter plot is plotted of emperical and theoritical distribution.

```
plt.scatter(deg_prob, exp_deg)
plt.show()
```

The scatter plot reveals that emperical and theoritical distribution are linearly correlated.

Finally, applying the Pearson R test on these distribution:

```
stats.pearsonr(deg_prob, exp_deg)
```

This outputs the correlation coefficient as 0.999985637 and p-value as `6.903125270534218e-154`. The small p-value indicates that the emperical distribution is indeed a binomial distribution.

Figure 7: Emperical vs Theoritical Distribution of Degrees

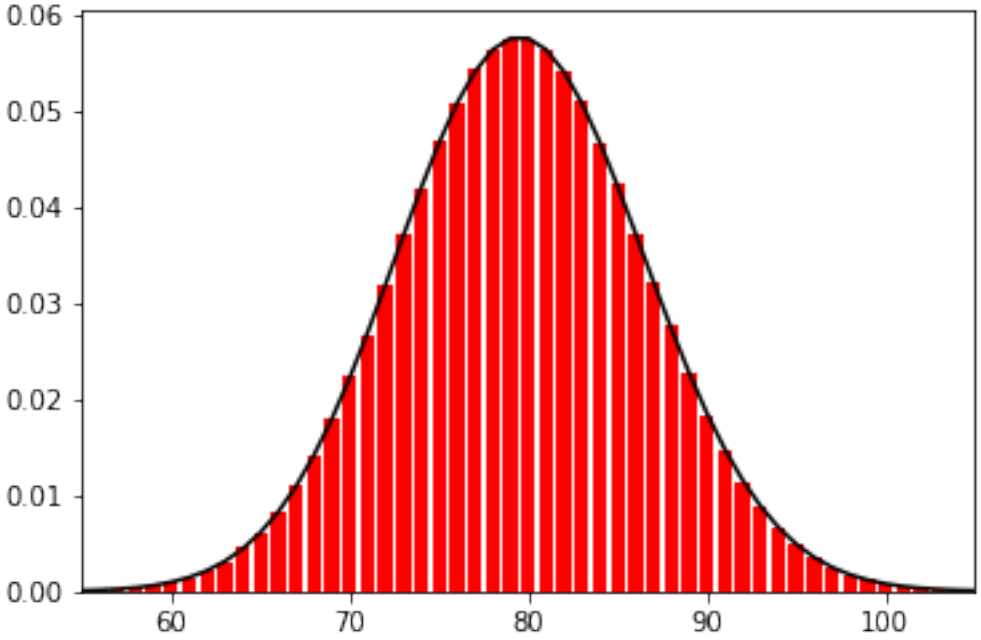
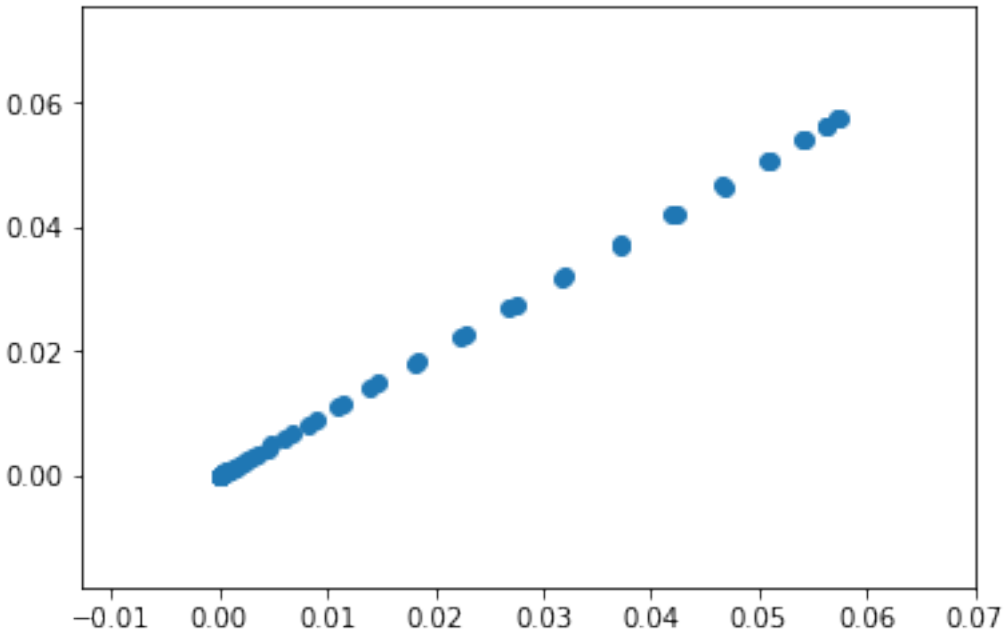


Figure 8: Correlation Plot of Emperical vs Theoritical Distribution of Degrees



3 Graph Metrics

3.1 Karate Club

For the Karate Club, the script was used to generate the CSV file (undirected graph), which was used as the input.

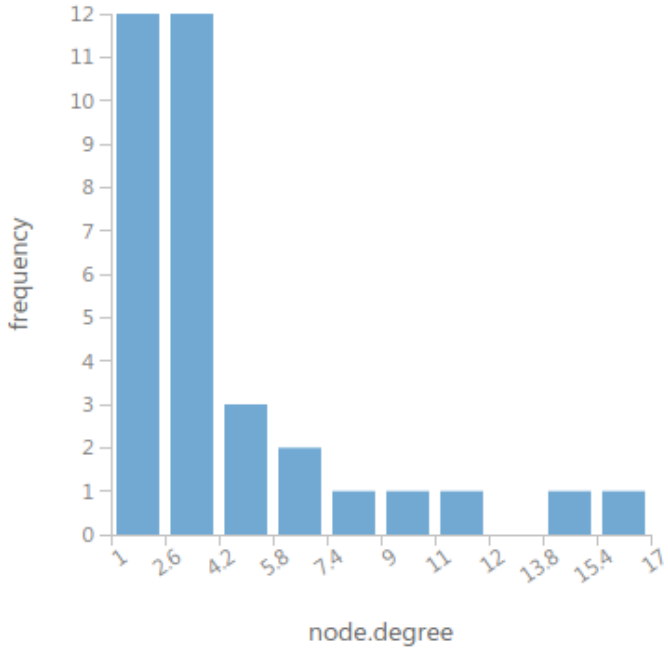


Figure 9: Histogram of degree frequencies

number.of.nodes	number.of.edges	max.degree	min.degree	avg.degree	diameter	avg.path.length	edge.density	largest.clique.size
34	77	17	1	4.529412	5	2.424242	0.137255	4

Figure 10: Other metrics

1. Histogram of degree frequencies: Shown in figure 9.
2. **Sparse Graph**: Number of edges are 77 and maximum possible edges are $\binom{34}{2}$. Thus graph density is $\frac{77}{\binom{34}{2}} = 0.13$. In computer sciences, programmmers are usually encouraged to implement specialized sparse graph based operations if edges are less than $V \lg V$, where V are number of vertices. In that case, this is a sparse graph.
3. **Network Diameter:** 5

4. **The average path length:** 2.4242

5. **The graphs average clustering coefficient:** 0.225743

3.2 Random Graph

For the random graph, the script was used to generate the CSV file which was used as the input in the Azure modules.

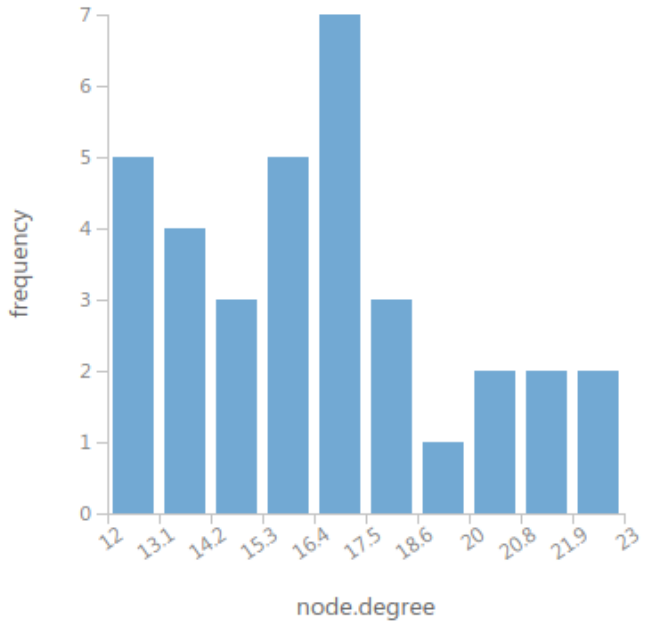


Figure 11: Histogram of degree frequencies

number.of.nodes	number.of.edges	max.degree	min.degree	avg.degree	diameter	avg.path.length	edge.density	largest.clique.size
34	282	23	12	16.588235	2	1.497326	0.502674	6

Figure 12: Other metrics

1. Histogram of degree frequencies: Shown in figure 11.

2. **Sparse Graph**: Number of edges are 282 and maximum possible edges are $\binom{34}{2}$. Thus graph density is $\frac{282}{\binom{34}{2}} = 0.5$. In computer sciences, programmmers are usually encouraged to implement specialized sparse graph based operations if edges are less than $V \lg V$, where V are number of vertices. In that case, this is a dense graph.

3. **Network Diameter:** 2
4. **The average path length:** 1.497326
5. **The graphs average clustering coefficient:** 0.503427

Notes

1. This given solution of the assignment follows the HKUST honour code. Although assignment has been discussed with other peers, the solutions are my own.
2. Kindly give detailed critical feedback on my assignment solutions. Your feedback is more important to us than grades.